```
TITLE "Ram control block";

%INCLUDE "temp.inc";%



%   Handle the handshaking between the rest of the system and the ram%
%   Neal Wilcer%
%   7/6/98%


SUBDESIGN ramctrl
(
    rfclk                           : INPUT;
    DivBy4                          : INPUT;    % Clock that synchs the ram R/Ws %
    RamSel                          : INPUT;    % Bit that selects the ram as the address and data source %
    VmeA[3..1]                      : INPUT;    % Address bits used to select ram bank %
    SeqMemOe                        : INPUT;    % Emu enabled signal from the emu register, if active (high)
                                                    enable the ram to be dumped to '374 latches %
    /reset                          : INPUT;    % reset signal for the entire board %
    RamAck                          : OUTPUT;   %Acknowledge from this block saying that the r/w was%
                                                %successful %
    Write                           : INPUT;    %Signal signifying a write of the ram%
    Read                            : INPUT;
    nBankWr                         : OUTPUT;   % Direction control for the bi-directional buffers that%
                                                %write or read data to/from the ram. Low = write (A > B)%
    nRIO_En[6..0]                   : OUTPUT;   % Enables for the bi-buffers. These signals are actually%
                                                %the lower 3 address bits %
    nBank_En[6..0]                  : OUTPUT;   % Enables for one of 7 banks of ram. These signals are also%
                                                %the lower 3 address bits %
    nRam_Write                      : OUTPUT;   % Write strobe to the ram. Must be a pulse%
    nRam_Oe                         : OUTPUT;   % Output enable for the ram. If low, the rams are set to pump%
                                                %or read data, if high, the rams are set to accept or have data %
                                                %written to them %

)
VARIABLE

    % used a gray code counter to eliminate glitches %
    ramaccess   : MACHINE
        OF BITS (st1,st2,st3)
        WITH STATES
            (NoOp           = B"000",
            CheckReadWrite  = B"001",
            ChooseRamBank   = B"011",
            PassData        = B"010",
            EndWrite        = B"110",
            SendAck         = B"111",
```

```
        CleanUp         = B"101",
        n8              = B"100"
        );

    AckFlop             :DFF;   % latch the RamAck signal %
    BankWrFlop          :DFF;   % latch the nBankWr signal %
    RIOFlop[6..0]       :DFF;   % latch the nRIO_en[6..0] signal %
    Bank_EnFlop[6..0]   :DFF;   % latch the nnBank_En[6..0] signal %
    RamWrFlop           :DFF;   % latch the nRam_Write signal %
    Ram_OeFlop          :DFF;   % latch the nRam_Oe signal %


BEGIN


    % List of signals used to set D inputs for flops that feed that outputs %
        AckFlop.d = (RamSel AND SendAck AND !SeqMemOe);
        BankWrFlop.d = !(Write AND !SeqMemOe AND RamSel);
        RIOFlop[0].d = !((vmea[3..1] == 0) AND !SeqMemOe AND (Write OR Read) AND RamSel);
        RIOFlop[1].d = !((vmea[3..1] == 1) AND !SeqMemOe AND (Write OR Read) AND RamSel);
        RIOFlop[2].d = !((vmea[3..1] == 2) AND !SeqMemOe AND (Write OR Read) AND RamSel);
        RIOFlop[3].d = !((vmea[3..1] == 3) AND !SeqMemOe AND (Write OR Read) AND RamSel);
        RIOFlop[4].d = !((vmea[3..1] == 4) AND !SeqMemOe AND (Write OR Read) AND RamSel);
        RIOFlop[5].d = !((vmea[3..1] == 5) AND !SeqMemOe AND (Write OR Read) AND RamSel);
        RIOFlop[6].d = !((vmea[3..1] == 6) AND !SeqMemOe AND (Write OR Read) AND RamSel);
        Bank_EnFlop[0].d = !((vmea[3..1] == 0) AND !SeqMemOe AND (Write OR Read) AND RamSel);
        Bank_EnFlop[1].d = !((vmea[3..1] == 1) AND !SeqMemOe AND (Write OR Read) AND RamSel);
        Bank_EnFlop[2].d = !((vmea[3..1] == 2) AND !SeqMemOe AND (Write OR Read) AND RamSel);
        Bank_EnFlop[3].d = !((vmea[3..1] == 3) AND !SeqMemOe AND (Write OR Read) AND RamSel);
        Bank_EnFlop[4].d = !((vmea[3..1] == 4) AND !SeqMemOe AND (Write OR Read) AND RamSel);
        Bank_EnFlop[5].d = !((vmea[3..1] == 5) AND !SeqMemOe AND (Write OR Read) AND RamSel);
        Bank_EnFlop[6].d = !((vmea[3..1] == 6) AND !SeqMemOe AND (Write OR Read) AND RamSel);
        RamWrFlop.d = !((Bank_EnFlop[] != 127) AND Write AND PassData AND !SeqMemOe AND RamSel );
        Ram_OeFlop.d = (Write AND !SeqMemOe AND (nRIO_En[] != H"7F"));


    % the clock used for the state machine %
    ramaccess.clk = DivBy4;

    AckFlop.clk = rfclk;
    BankWrFlop.clk = rfclk;
    Ram_OeFlop.clk = rfclk;
    RIOFlop[].clk = rfclk;
    Bank_EnFlop[].clk = rfclk;
    RamWrFlop.clk = rfclk;

    % power up/ reset conditions %
```

```
AckFlop.clrn = /reset;
BankWrFlop.prn = /reset;
Ram_OeFlop.clrn = /reset;
RIOFlop[].prn = /reset;
Bank_EnFlop[].prn = /reset;
RamWrFlop.prn = /reset;

RamAck = AckFlop.q;
nBankWr = BankWrFlop.q;
nRIO_En[6..0] = RIOFlop[6..0].q;
nBank_En[6..0] = Bank_EnFlop[6..0].q;
nRam_Write = RamWrFlop.q;
nRam_Oe = Ram_OeFlop.q;




    CASE ramaccess IS
        WHEN NoOp =>    % wait for the ram select bit to be active before we do anything %
            IF !RamSel THEN
                ramaccess = NoOp;
            ELSE
                ramaccess = CheckReadWrite;
            END IF;
        WHEN CheckReadWrite =>  % wait here until either the read or write line goes active %
            IF ((Write == GND) AND (Read == GND)) THEN
                ramaccess = CheckReadWrite;
            ELSE
                ramaccess = ChooseRamBank;
            END IF;
        WHEN ChooseRamBank =>   % if the lower three bits of the vme address == 7, the user is trying to
                                    access a non existant bank of ram, so just send a dtack  %
            CASE (VmeA[]) IS
                WHEN 7 =>
                    ramaccess = SendAck;
                WHEN OTHERS =>
                    ramaccess = PassData;
            END CASE;
        WHEN PassData =>
            IF ((Write == VCC) OR (Read == VCC))THEN
                ramaccess = EndWrite;
            ELSE
                ramaccess = SendAck;
            END IF;
        WHEN EndWrite =>
            ramaccess = SendAck;
        WHEN SendAck =>
```

```
            IF (RamSel) THEN    % if ramsel is still active , wait here until it goes inactive %
                ramaccess = SendAck;
            ELSE
                ramaccess = CleanUp;
            END IF;
        WHEN CleanUp =>
            ramaccess = n8;
        WHEN n8 =>
            ramaccess = NoOp;
    END CASE;


END;
```